INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Model-Checking of CTL on Infinite Kripke Structures Defined by Simple Graph Grammars

Yves-Marie Quemener and Thierry Jéron

N° 2563

Juin 1995

PROGRAMME 1

ISSN 0249-6399

Model-Checking of CTL on Infinite Kripke Structures Defined by Simple Graph Grammars

Yves-Marie Quemener and Thierry Jéron*

Programme 1 — Architectures parallèles, bases de données, réseaux
et systèmes distribués
Projet Pampa

Rapport de recherche n ° 2563 — Juin 1995 — 16 pages

Abstract: We present an algorithm for checking whether an infinite transition system, defined by a graph grammar of a restricted kind, is a model of a formula of the temporal logic CTL. We first present the syntax and the semantics of CTL, that are defined with respect to transition systems, labelled with atomic propositions. Then, we show how to adapt the formalism of graph grammars, for expressing such infinite transition systems. Our algorithm treats such a finite representation, and modify it, ensuring that the labelling for formulas remains coherent with the truth values of the different states of the infinite transition system.

Key-words: model-checking, CTL, verification, infinite-state systems, graph grammars

(Résumé : tsvp)

*IRISA, Campus de Beaulieu, F-35042 Rennes, FRANCE. E-mail: name@irisa.fr

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu, 35042 RENNES Cedex (France)
Téléphone : (33) 99 84 71 00 – Télécopie : (33) 99 84 71 71

Model-checking de CTL sur des structures de Kripke infinies définies par des grammaires de graphes simples

Résumé : Nous présentons un algorithme qui détermine si un système de transition infini, défini par une grammaire de graphes d'un type restreint, est ou non un modèle d'une formule de la logique temporelle CTL. Nous présentons d'abord la syntaxe et la sémantique de CTL, qui sont définies suivant des systèmes de transitions étiquetés par des propositions atomiques. Puis, nous montrons comment adapter le formalisme des grammaires de graphes pour exprimer de tels systèmes de transition infinis. Notre algorithme travaille sur ce type de représentation finie, et la modifie de manière à assurer que l'étiquetage par les formules demeure cohérent avec les valeurs de vérité des différents états du système de transition infini.

Mots-clé : model-checking, CTL, vérification, systèmes d'état infini, grammaire de graphes

1 Introduction

Model-checking is a widely used method for the verification of distributed systems. The overall behaviour of a distributed system is modeled as a transition system, whose states represent the global states of the distributed system, and whose transition relation gives the possible evolutions of the system. It can be checked whether such a transition system is a model of a temporal logic formula. Until recently, model-checking was only used for finite-state models. But, Muller and Schupp [MS85] showed that the pushdown-automata transitions graphs have a decidable monadic second-order theory. Courcelle, [Cou89], extended that result, by showing that equational graphs also have a decidable second-order theory, since Caucal, [Cau92], showed that the pushdown-automata transition graphs are exactly the rooted, finite-degree, equational graphs. Hence, some infinite-state systems can have finite representations, which can be used for different verification methods. For example, Burkart and Steffen, [BS94], present an algorithm for the model-checking of the alternation-free mu-calculus on pushdown processes. Methods published so far study *context-free* or *pushdown processes*, i.e. restricted process algebras such that the infinite transition graphs of their terms are pushdown-automata transition graphs.

We are interested in the model-checking of *communicating finite-state systems* (for short CFSMs), i.e. distributed systems described as finite-state automata communicating by messages through FIFO queues. Jéron has presented a semi-decision procedure of the infinity of the transition systems representing CFSMs [Jér93]. That procedure detects certain sequences that will be infinitely repeated. Extending that procedure, we are currently developing a test which will enable us to extract, for certain CFSMs, a graph grammar representing the infinite transition system of a CFSM. Hence, we are interested in algorithms that perform model-checking of infinite graphs defined by graph grammars. We present here a first algorithm of that kind, which make a model-checking of the temporal logic CTL [CES86] on a graph defined by a simple kind of graph grammar.

We first present the temporal logic CTL, and then we show how to use certain graph grammars, of a simple kind, to define certain infinite structures, which can be used as a model of distributed systems. CTL formulas are inter-

preted on those structures, and we then present our model-checking algorithm. That algorithm has to modify the finite representation of an infinite transition system, for ensuring that the labelling of states of the finite representation by subformulas of the formula being checked remains coherent with the truth values for the states of the infinite structure. One important point is to show that it is necessary to explore only a finite part of the infinite transition system for deciding whether a state satisfies a formula or not.

2 The temporal logic CTL

CTL is a branching-time temporal logic, which is interpreted with respect to states of transitions systems. Those states are labelled with atomic propositions. The CTL formulas can be expressed as formulas of the mu-calculus without alternation of fixpoints. They can express properties of safety or liveness, but not of fairness.

2.1 Syntax

AP is the underlying set of atomic propositions. The formal syntax for CTL formulas is:

- Every atomic proposition $p \in AP$ is a CTL formula.
- If φ_1 and φ_2 are CTL formulas, then so are $\neg\varphi_1$, $\varphi_1 \wedge \varphi_2$, $\forall\circ\varphi_1$, $\exists\circ\varphi_1$, $\forall(\varphi_1 \mathcal{U} \varphi_2)$, $\exists(\varphi_1 \mathcal{U} \varphi_2)$.

tt is the atomic proposition which is true in all states. The following syntactical definitions are added:

- $\forall(tt\mathcal{U}\varphi)$ is written $\forall\Diamond\varphi$, and $\exists(tt\mathcal{U}\varphi)$ is written $\exists\Diamond\varphi$.
- $\neg\forall\Diamond(\neg\varphi)$ is written $\exists\Box\varphi$, and $\neg\exists(\neg\varphi)$ is written $\forall\Box\varphi$.

Informally, the temporal connectives can be described as follows: $\forall\circ\varphi$ (resp. $\exists\circ\varphi$) is true of a state if all (resp. at least one of) its successors satisfy φ; $\forall(\varphi_1 \mathcal{U} \varphi_2)$ (resp. $\exists(\varphi_1 \mathcal{U} \varphi_2)$) is true of a state s if on all (resp. on one of) the paths issued from s, the states verify φ_1 *until* one state satisfy φ_2.

Hence, if a state satisfy $\forall\Diamond\varphi$, it means that, in the "future" of that state, a state which satisfy φ is unavoidable; $\exists\Diamond\varphi$ means that it is possible to reach a state which satisfy φ. Similarly, a state satisfy $\forall\Box\varphi$, if all the states reachable from that state satisfy φ; and a state satisfy $\exists\Box\varphi$ if there is a path issued from that state on which all states satisfy φ.

For each formula φ of CTL, we note $sf(\varphi)$ the set of all subformulas of φ, including φ itself.

2.2 Semantics

The semantics of CTL formulas is defined with respect to a *Kripke structure*. A Kripke structure is a triple $K = (S, R, L)$ where:

- S is a set of states.
- R is a binary relation on S ($R \subseteq S \times S$), which is the transition relation. R must be *total*, i.e. $\forall s \in S, \exists t \in S, (s,t) \in R$.
- $L : S \rightarrow 2^{AP}$ assigns to each state the set of atomic propositions which are true in that state.

A *path* is a sequence of states $(s_0, s_1, \ldots)$ such that $\forall i, (s_i, s_{i+1}) \in R$. The notation $K, s_0 \models \phi$ means that the formula ϕ holds at state s_0 in the structure K. We drop K when it's obvious from the context which structure is concerned. The relation $\models$ is defined inductively as follows:

- $s_0 \models p$ iff $p \in L(s_0)$.
- $s_0 \models \neg\varphi$ iff not$(s_0 \models \varphi)$.
- $s_0 \models \varphi_1 \wedge \varphi_2$ iff $s_0 \models \varphi_1$ and $s_0 \models \varphi_2$.
- $s_0 \models \exists\circ\varphi$ iff $\exists t \in S, (s_0, t) \in R$ and $t \models \varphi$.
- $s_0 \models \forall\circ\varphi$ iff $\forall t \in S, (s_0, t) \in R \Rightarrow t \models \varphi$.
- $s_0 \models \exists(\varphi_1 \mathcal{U} \varphi_2)$ iff for some path $(s_0, s_1, \ldots)$,
  $\exists i[0 \leq i < \infty \wedge s_i \models \varphi_2 \wedge \forall j[0 \leq j < i \Rightarrow s_j \models \varphi_1]]$.
- $s_0 \models \forall(\varphi_1 \mathcal{U} \varphi_2)$ iff for all paths $(s_0, s_1, \ldots)$,
  $\exists i[0 \leq i < \infty \wedge s_i \models \varphi_2 \wedge \forall j[0 \leq j < i \Rightarrow s_j \models \varphi_1]]$.

3 Structures defined by graph grammars

3.1 Informal presentation

Certain infinite graphs can be finitely described by graphs grammars. Informally, those graphs are composed of a finite number of patterns that are infinitely compounded in a regular way.

For expressing the gluing of patterns, it is convenient to use hyperarcs, which are arcs linking one or more states, instead of two for classical arcs. To each labelled hyperarc is associated a graph; the states of the hyperarc indicate where the gluing must be made. Those rules, associating an hyperarc and a finite graph, for compounding patterns compose a graph grammar. The applications of the rules of a graph grammar to an initial axiom graph determines a unique infinite graph when the grammar is deterministic, *i.e.* when there is only one rule for each kind of hyperarc.

We restrict ourselves to simple infinite graphs. The graphs we study are made of an initial graph, and of the compounding of one kind of pattern in a linear way, *i.e.* only one pattern is compounded to a precedent pattern.

3.2 Definitions

We use a formalism similar to that of graph grammars for finitely expressing certain infinite structures. We adapt it to cope with the labelling of states with atomic propositions. Our presentation is inspired with the presentation of graph grammars made in [Cau92].

Definition: A *graded alphabet* F is a finite set of letters, where a positive integer is associated with each letter $f \in F$. That integer is the *arity* of f. F can be partitionned in sets F_i, with $f \in F_i$ iff the arity of f is i.

Definition: Let F be a graded alphabet. An *hyperstructure* $\overline{K}$ is a couple (K, H_K) where $K = (S_K, R_K, L_K)$ is a finite Kripke structure, and H_K is a finite set of *hyperarcs* $fs_1 \dots s_n$. The hyperarc $fs_1 \dots s_n$ is labelled by the *non-terminal* $f \in F_n$ and the s_i are distinct states of S_K.

The hyperarcs added to the classical Kripke structure will enable us to indicate where the gluing of other Kripke structures must be made.

Definition: A *structure grammar* $\mathcal{G}$ is a finite set of hyperarc replacement rules

$fs_1 \dots s_n \rightarrow \overline{K}$, where $fs_1 \dots s_n$ is an hyperarc, and $\overline{K}$ is an hyperstructure, and the s_i are distinct states of S_K.

The rules of such a grammar indicate what pattern has to be glued to an initial hyperstructure, and the states s_i indicate which states have to be merged for gluing the pattern. This gluing operation is precisely defined as follows.

Definition: Given a structure grammar $\mathcal{G}$ and an hyperstructure $\overline{M}$, $\overline{M}$ *rewrites in one step* to a hyperstructure $\overline{N}$, and we note $\overline{M} \rightarrow_{\mathcal{G}} \overline{N}$, if, for some rule $(fs_1 \dots s_n \rightarrow \overline{K}) \in \mathcal{G}$, we have:

- $\exists (t_1, \dots, t_n) \in (S_M)^n : ft_1 \dots t_n \in H_M$
- $\forall i \in \{1, \dots, n\} : L_K(s_i) = L_M(t_i)$
- and, for some matching function g mapping s_i to t_i, and mapping injectively the other states of S_K to states outside of S_M:
  - $H_N = (H_M - \{ft_1 \dots t_n\}) \cup \{fg(s_1) \dots g(s_n), fs_1 \dots s_n \in H_K\}$
  - $S_N = S_M \cup \{g(s), s \in S_K\}$
  - $R_N = R_M \cup \{(g(s), g(t)), (s,t) \in R_K\}$
  - $L_N : S_N \rightarrow 2^A P$ is defined as follows; $L_N(s) = L_M(s)$ if $s \in S_M$ or $L_N(s) = L_K(t)$ if $s = g(t)$ for some $t \in S_K$

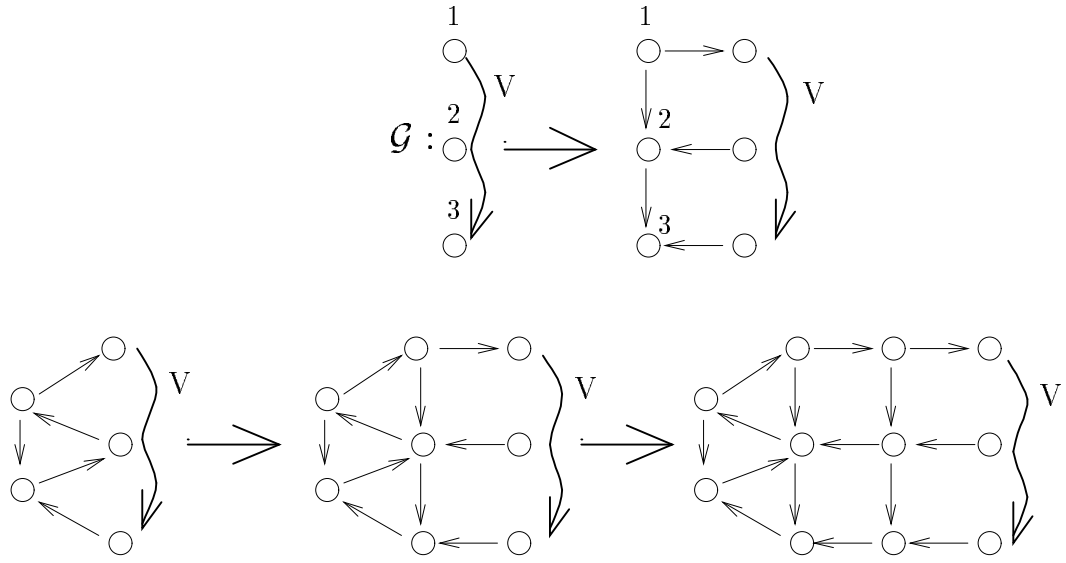
When an hyperstructure rewrites in another, some states of the initial structure have to be merged with states of the pattern. We impose that the labelling of those merged states is the same, hence we can define unambiguously the labelling of the states of the resulting hyperstructure.

The figure on the following page presents a graph grammar $\mathcal{G}$, composed of one rule, that uses the hyperarc $V123$, and two successive rewritings according to that rule.

From now on, we restrict our study to simple structure grammars, i.e. grammars with only one rule, whose right member has only one hyperarc.

Let N be a natural, and V a label of hyperarc of arity N.

Let $G_0 = (S_{G_0}, R_{G_0}, L_{G_0})$ be a finite Kripke structure with N distinguished, pairwise distinct, states, $(ex_i)_1^N$, and $A = (S_A, R_A, L_A)$ a finite Kripke structure



with $2N$ distinguished, pairwise distinct states, $(in_i)_1^N$ and $(out_i)_1^N$. We further impose that: $\forall i \in [1..N], L_{G_0}(ex_i) = L_A(in_i) = L_A(out_i)$.

Let $\overline{G_0} = (G_0, H_{G_0} = \{Vex_1 \dots ex_N\})$, and $\overline{A} = (A, H_A = \{Vout_1 \dots out_N\})$; and let $\mathcal{G}_0$ be the structure grammar with the unique rule $Vin_1 \dots in_N \rightarrow \overline{A}$.

We show that the infinite iterative application of the unique rule of $\mathcal{G}_0$ on the hyperstructure $\overline{G_0}$ determines a unique infinite Kripke structure $K = (S, R, L)$.

Proof: First, we show that the unique rule of $\mathcal{G}_0$ can be infinitely applied on $\overline{G_0}$.

Since we have $Vex_1 \dots ex_N \in H_{G_0}$, and $\forall i \in [1..N], L_{G_0}(ex_i) = L_A(in_i)$, we can apply the unique rule of $\mathcal{G}_0$ to $\overline{G_0}$. We have: $\overline{G_0} \rightarrow_{\mathcal{G}_0} \overline{G_1}$, and we call g_1 the matching function associated to that rewriting step.

Since $H_A = \{Vout_1 \dots out_N\}$, we have $H_{G_1} = \{Vg_1(out_1) \dots g_1(out_N)\}$, and, $\forall i \in [1..N], L_{G_1}(g_1(out_i)) = L_A(out_i) = L_A(in_i)$.

We suppose that there exists j hyperstructures $\overline{G_1}, \overline{G_2}, \dots, \overline{G_j}$ such that:

- $\overline{G_0} \rightarrow_{\mathcal{G}_0} \overline{G_1} \rightarrow_{\mathcal{G}_0} \overline{G_2} \rightarrow_{\mathcal{G}_0} \dots \rightarrow_{\mathcal{G}_0} \overline{G_j}$

- $\exists (v_1, \ldots, v_N) \in S^N_{G_j}, H_{G_j} = \{V v_1 \ldots v_N\} \wedge \forall i \in [1..N], L_{G_j}(v_i) = L_A(in_i)$

Then, in the same way as for $\overline{G_0}$, we can apply the rule of $\mathcal{G}_0$ to $\overline{G_j}$, which rewrites in one step to $\overline{G_{j+1}}$. The hyperstructure $\overline{G_{j+1}}$ verifies the conditions that are necessary to apply one more time the rule of $\mathcal{G}_0$ to $\overline{G_{j+1}}$. We construct in that way an infinite set of hyperstructures $\overline{G_j}$, for all positive integers j.

We now define $K = (S, R, L)$:

- $S = \cup S_{G_j}$, for all positive integers j.

- $R = \{(s,t), \exists j, (s,t) \in R_{G_j}\}$.

- $\forall s \in S, L(s) = L_{G_j}(s)$, for a j such that $s \in S_{G_j}$. □

Definition: We say that (G_0, A) is a *finite representation* of K.

We call g_i the matching function associated to the ith rewriting step of the grammar $\mathcal{G}_0$ on $\overline{G_0}$, and g_0 the identity on S_{G_0}.

We have: $\forall s \in S : \exists t \in (S_{G_0} \cup S_A), \exists i, s = g_i(t)$

In general, t is not unique. We call $rep(s)$ the set of states $t \in (S_{G_0} \cup S_A)$ such that we have: $\exists i, s = g_i(t)$.

We distinguish different categories of states of S for making easier the presentation of the algorithm.

Definition: We say that $s \in S$ is an *inner-state* of the i-expansion, (resp. of the 0-expansion), iff $\exists t \in S_A - \{in_1, \ldots, in_N, out_1, \ldots, out_N\}, s = g_i(t)$, (resp. $\exists t \in S_{G_0} - \{ex_1, \ldots, ex_N\}, s = g_0(t)$).

Definition: We say that $s \in S$ is a *frontier-state* of the i-expansion, (resp. of the 0-expansion), iff $\exists j \in [1..N], s = g_i(out_j) = g_{i+1}(in_j)$, (resp. $\exists j \in [1..N], s = g_0(ex_j) = g_1(in_j)$).

Let (G, B) be a finite representation of an infinite structure K. Let φ be a CTL formula. Let L^φ be a labelling function on the states of G and B, for all the subformulas of φ; $L^\varphi : (S_G \cup S_B) \times sf(\varphi) \rightarrow \{True, False\}$.

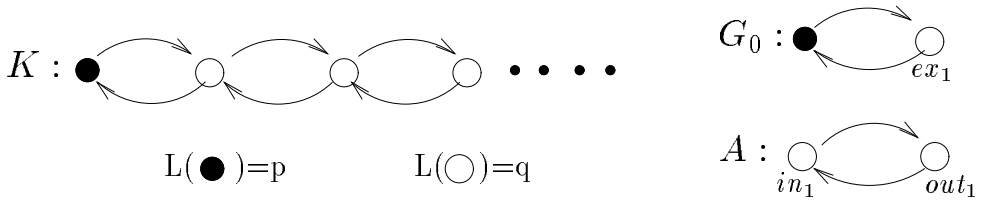
Definition: We say that (G, B), supplied with L^φ, is *coherent* for φ with K iff

$$\forall s \in S, \forall \varphi_0 \in sf(\varphi) : s \models \varphi_0 \Leftrightarrow \forall t \in rep(s), L^\varphi(t, \varphi_0) = True$$

For all $p \in AP$, we define L^p: $\forall s \in S_{G_0} \cup S_A, L^p(s,p) = True$ iff $s \in S_{G_0} \wedge p \in L_{G_0}(s)$ or $s \in S_A \wedge p \in L_A(s)$. Using these labellings, we have, by

construction, that (G_0, A) is a finite representation, coherent for the atomic propositions with K.

The following figure gives an example of such a finite representation.



In that case, the hyperarc which indicates the gluing of patterns includes only one state. The labelling by atomic propositions distinguish only two sets of states. The first set is composed of the unique state at the extremity of the infinite structure, the other set is composed of all the other states.

4 The algorithm

4.1 Overview of the algorithm

A linear algorithm for checking CTL formulas on finite structures was presented in [CES86]. This algorithm operates as follows: a finite structure is given; all states will be labelled by all subformulas of φ which are true in them, working incrementally, beginning with the simplest subformulas. This will ensure that, each time the algorithm has to verify a given formula, the truth value of all the pertinent subformulas will be known for all states of the structure.

Our algorithm for checking infinite Kripke structures uses the same principle. A finite representation of an infinite structure is given, with a labelling coherent for the atomic propositions. We progress incrementally, deciding the labelling of more and more complex subformulas for the states of our finite representation. But, that representation must remain coherent with the infinite structure for all subformulas.

For boolean connectives, the truth value of a given state only depends on the truth value of the subformulas at the same state. Hence, for example, a

coherent representation for the subformulas φ_1 and φ_2 will remain coherent for the formula $\varphi_1 \wedge \varphi_2$, if we define as follows the labelling $L^{\varphi_1 \wedge \varphi_2}$:

- $\forall s \in S_{G_0} \cup S_A, \forall \varphi_1' \in sf(\varphi_1), L^{\varphi_1 \wedge \varphi_2}(s, \varphi_1') = L^{\varphi_1}(s, \varphi_1')$
- $\forall s \in S_{G_0} \cup S_A, \forall \varphi_2' \in sf(\varphi_2), L^{\varphi_1 \wedge \varphi_2}(s, \varphi_2') = L^{\varphi_2}(s, \varphi_2')$
- $\forall s \in S_{G_0} \cup S_A, L^{\varphi_1 \wedge \varphi_2}(s, \varphi_1 \wedge \varphi_2) = True \Leftrightarrow L^{\varphi_1}(s, \varphi_1) = True \wedge L^{\varphi_2}(s, \varphi_2) = True$

This is no more true for temporal connectives, because the truth value of a temporal formula for a state depends on the truth value of subformulas for other states. We will then have to modify our finite representation.

For the $\forall \circ(\varphi)$ and $\exists \circ(\varphi)$ formulas, the states of the infinite structure, represented by one state of A, have all the same immediate successors, except when they are given by the first expansion of the grammar. In that case, the in-states of A are merged with the ex-states of the initial graph G_0, and they can have immediate successors in G_0, whereas, for the following expansions, the in-states are merged with the out-states of the preceding expansion, and their immediate successors have all the same labelling for the subformula φ. Therefore, it may be necessary to modify the representation. This is simply achieved by taking as the new initial graph the merging of G_0 and A.

For the $\exists(\varphi_1 \mathcal{U} \varphi_2)$ and $\forall(\varphi_1 \mathcal{U} \varphi_2)$ formulas, the modification will be more complicated, because the truth value of a state for such a formula can depend on all its successors, and not only the immediate ones. We present now the algorithm for $\exists \varphi_1 \mathcal{U} \varphi_2$ formulas. The algorithm for $\forall(\varphi_1 \mathcal{U} \varphi_2)$ is very similar and is not presented here.

4.2 The algorithm for $\exists(\varphi_1 \mathcal{U} \varphi_2)$ formulas

Let (G_0, A) be a finite representation of an infinite structure $K = (S, R, L)$, with labellings L^{φ_1} and L^{φ_2}, coherent with φ_1 and φ_2. We show here how to obtain a new representation (G_0', A'), and a labelling $L^{\exists \varphi_1 \mathcal{U} \varphi_2}$, coherent with $\exists(\varphi_1 \mathcal{U} \varphi_2)$. We begin by showing three lemmas, that enable us to restrict the search of a new representation to the study of the frontier-states, and then we present our algorithm.

4.2.1 Three lemmas

Lemma 1: We can know for sure that an inner-state s of the i-expansion satisfies $\exists(\varphi_1 \mathcal{U} \varphi_2)$, if we know if the frontier-states of the $(i-1)$-expansion and of the i-expansion satisfy $\exists(\varphi_1 \mathcal{U} \varphi_2)$.

Proof: All the paths issued from s are first composed of the inner-states of the i-expansion. They are composed of other states only after they include the frontier-states of the $(i-1)$- or i-expansion. Hence, if we know if the frontier states of the $(i-1)$- and i- expansion satisfy $\exists(\varphi_1 \mathcal{U} \varphi_2)$, we don't need to study the paths issued from s, which can include an infinite number of states, beyond those frontier states. □

Nota bene: For the inner-states of the 0-expansion, *i.e.* the states of G_0, except the states (ex_i), it is enough to know if the frontier-states of the 0-expansion , *i.e.* the states (ex_i), satisfy $\exists(\varphi_1 \mathcal{U} \varphi_2)$.

Lemma 2: We can know for sure that a frontier-state $s = g_i(out_j)$, with $i \geq 1$, of the i-expansion satisfies $\exists(\varphi_1 \mathcal{U} \varphi_2)$, if we know if the frontier-states of the $(i-1)$-expansion satisfy $\exists(\varphi_1 \mathcal{U} \varphi_2)$, by studying the labelling for φ_1 and φ_2 of the inner- and frontier- states of the $(i+1)$-, $(i+2)$-, ..., $(i+N)$-expansions, where N is the number of states (ex_j), (in_j) or (out_j).

Proof: If we find in A a path $(out_j = s_0, s_1, \ldots, s_n)$ with $L^{\varphi_2}(s_n, \varphi_2) = True$ and $L^{\varphi_1}(s_k, \varphi_1) = True$, $\forall k, 0 \leq k < n$, we can conclude that $s = g_i(out_j) \models \exists(\varphi_1 \mathcal{U} \varphi_2)$, since we know that there is in K a path $(g_i(out_j), g_i(s_1), \ldots, g_i(s_n))$, with $g_i(s_n) \models \varphi_2$ and $\forall k, 0 \leq k < n, g_i(s_k) \models \varphi_1$, since the labellings L^{φ_1} and L^{φ_2} are coherent with φ_1 and φ_2. We can conclude the same if we find such a path in A, issued from in_j, since $s = g_i(out_j) = g_{i+1}(in_j)$.

Let suppose we don't find any of those paths. If we find in A a path $(out_j = s_0, s_1, \ldots, s_n = in_k)$ with $L^{\varphi_1}(s_i, \varphi_1) = True, \forall i, 0 \leq i \leq n$, we can decide if that path enables us to conclude that $s \models \exists(\varphi_1 \mathcal{U} \varphi_2)$, since we know whether the frontier-state of the $(i-1)$-expansion $g_{i-1}(out_k) = g_i(in_k)$ satisfies $\exists(\varphi_1 \mathcal{U} \varphi_2)$ or not. But, if we find in A such a path from in_j to out_k, it means that we have in K a path from $g_i(out_j) = g_{i+1}(in_j)$ to $g_{i+1}(out_k)$, and, on all the states of that path, φ_1 holds. Hence, we have to make a new exploration of the pattern A, beginning at out_k and in_k. But, if $j = k$, we have already made that exploration. Similarly, if there is a path from in_k to out_l, and $l = j$

or $l = k$, we don't need to study the paths in A issued from in_l or out_l. Hence, it is necessary to study at most N expansions after the i-expansion. □

Nota bene: For the frontier-states of the 0-expansion, the exploration of G_0 stays for the exploration of A which stops at the frontier-states of the precedent expansion.

Lemma 3: Suppose there exists i and j such that:

$$\forall k \in [1 \dots N], g_i(out_k) \models \exists(\varphi_1 \mathcal{U} \varphi_2) \Leftrightarrow g_j(out_k) \models \exists(\varphi_1 \mathcal{U} \varphi_2)$$

Then, we have:

$$\forall k \in [1 \dots N], g_{i+1}(out_k) \models \exists(\varphi_1 \mathcal{U} \varphi_2) \Leftrightarrow g_{j+1}(out_k) \models \exists(\varphi_1 \mathcal{U} \varphi_2)$$

Proof: This is a direct consequence of the lemma 2. The labellings for φ_1 and φ_2 are the same for the states of the N following expansions after the $(i+1)$- and $(j+1)$-expansions, since they are the labellings L^{φ_1} and L^{φ_2} of A. Hence, the only differences which can arise for the truth values for $\exists(\varphi_1 \mathcal{U} \varphi_2)$ of the frontier-states of the $(i+1)$- and $(j+1)$-expansions are those between the frontier-states of the i- and j-expansions. If they are identical, the truth values of the following frontier-states will be identical. □

4.2.2 Details of the algorithm

Our algorithm is decomposed in the following steps:

First step: We make a model-checking of the finite structures G_0 and A, that gives a first truth value of $\exists(\varphi_1 \mathcal{U} \varphi_2)$ for the ex-states of G_0 and the in-states and out-states of A. Then, we seek the paths in G_0, on which φ_1 is always true, that connect an ex-state to another ex-state, and the similar paths in A from an in/out-state to another in/out-state. This can be done by model-checkings of the formulas $\exists \varphi_1 \mathcal{U}(\varphi_1 \wedge p_i)$, where p_i is an atomic proposition which is only true of a given ex/in/out-state. Hence, we need to make N model-checkings of G_0 and $2N$ model-checkings of A.

Second step: The results of the first step will enable us to determine if the frontier-states of the different expansions satisfy $\exists(\varphi_1 \mathcal{U} \varphi_2)$. The temporary truth values given by the first step indicate whether there is a path, in the immediate neighbour patterns of a given frontier-state, on which φ_1 holds,

which leads to a state on which φ_2 holds. More precisely, if we call $temp(s_j)$ the temporary truth value obtained for an ex-, in-, or out-state, we have:

- $temp(ex_j) = True \Rightarrow ex_j \models \exists(\varphi_1 \mathcal{U} \varphi_2)$
- $temp(in_j) = True \Rightarrow \forall i, 0 \leq i, g_i(in_j) \models \exists(\varphi_1 \mathcal{U} \varphi_2)$
- $temp(out_j) = True \Rightarrow \forall i, 0 < i, g_i(out_j) \models \exists(\varphi_1 \mathcal{U} \varphi_2)$

But, frontier-states can also satisfy $\exists(\varphi_1 \mathcal{U} \varphi_2)$ because of the existence of a path, on which φ_1 holds, which "goes through" a pattern A. Those paths are detected in the first step of the algorithm. For example, if there is such a path from in_j to out_k, and $temp(out_k) = True$, we can conclude that $\forall i, 0 \leq i, g_i(in_j) \models \exists(\varphi_1 \mathcal{U} \varphi_2)$.

According to lemma 2, it is sufficient to study at most N expansions after a given expansion for deciding whether the frontier-states of that expansion satisfy or not $\exists(\varphi_1 \mathcal{U} \varphi_2)$. Hence, we can decide it for the frontier-states of the 0-expansion in the following way: we build a $N \times N$ matrix graph with the temporary truth values for the frontier-states of the first expansions, and we propagate the positive truth values backwards the links between frontier states which have been determined at the first step.

As soon as we have the definitive truth values for the frontier-states of the 0-expansion, we can use a similar procedure, according to lemma 2, to determine successively the truth values of the successive expansions. We iterate that procedure until we detect that the frontier-states of the p-expansion and of the q-expansion have the same definitive truth values, with $p < q$. This iteration is bound to terminate since there is at most 2^N possible combinations of the N definitive truth values of the frontier-states of a given expansion.

According to lemma 3, and by iterating it for the successive sets of frontier-states, we have then:

$$\forall i, 0 \leq i, \forall j \in [1..N], g_{p+i}(out_j) \models \exists(\varphi_1 \mathcal{U} \varphi_2) \Leftrightarrow g_{q+i}(out_j) \models \exists(\varphi_1 \mathcal{U} \varphi_2)$$

In other words, after the frontier-states of the first p expansions, the definitive truth values of the frontier-states of the following expansions will be regularly repeated according to a pattern of size $q - p$. According to lemma 1, the same holds for the inner-states. It is then direct that a finite representation (G'_0, A'), coherent for $\exists(\varphi_1 \mathcal{U} \varphi_2)$ with K, can be defined. G'_0 is obtained by

applying p times the rule of $\mathcal{G}_0$ to G_0. A' is obtained by merging $q-p$ patterns A, beginning at the p-expansion.

Third step: We know the definitive truth values of the ex-, in-, and out-states of the coherent representation (G'_0, A'). We can now make a model-checking of $\exists(\varphi_1 \mathcal{U} \varphi_2)$ for all the states of that representation. According to lemma 1, that will give us a coherent labelling $L^{\exists(\varphi_1 \mathcal{U} \varphi_2)}$.

4.3 Complexity

Let $|G_0|$ and $|A|$ be the sizes of the two parts of a finite representation of a Kripke structure K, and $|\varphi|$ the length of the CTL formula we check on K. For model-checking φ, the algorithm will determine $|\varphi|$ different labellings of the finite representation, as it processes upwards, and computes more and more complex subformulas. We call $mod(\varphi)$ the number of $\forall\mathcal{U}$ and $\exists\mathcal{U}$ connectives in φ. The eventual modifications of the finite representation can make grow its overall size up to $|G_0| + |A|2^{Nmod(\varphi)}$.

We overvalue the complexity of our algorithm by taking as the initial size of our finite representation $|G_0| + |A|2^{Nmod(\varphi)}$. We again overvalue by taking for all $|\varphi|$ steps the complexity of the steps for $\forall\mathcal{U}$ and $\exists\mathcal{U}$ connectives.

The first step of the algorithm is decomposed in a model-checking of G_0 and A, followed by N model-checkings of G_0 and $2N$ model-checkings of A for finding the paths "going through" patterns.

The second step is composed of at most 2^N model-checkings of a graph of size N^2.

The third step is the definitive model-checking of the finite representation.

Hence the overall complexity is in $\mathcal{O}(|\varphi|[(N+2)|G_0| + ((N+1)|A| + N)2^{Nmod(\varphi)+1}])$.

5 Conclusion

The algorithm we have presented is a first attempt at model-checking infinite-state systems which can be finitely represented by graph grammars. It has been implemented as a Pascal program. However, it is still too limited. The algorithm that we are currently developing is designed for extracting more general graph grammars, of the following kind: an initial graph "calls" a first

pattern A_1, which "calls" itself and an other pattern A_2, and so on, up to a pattern A_p, which only "calls" itself. We will then have to extend our algorithm for being able to cope with such grammars.

Other interesting research directions will be to try to extend the logic checked, and to use other methods of model-checking, like the local model-checking, or the symbolic model-checking.

References

[BS94] O. Burkart and B. Steffen. Pushdown Processes: Parallel Composition and Model Checking. In *CONCUR '94, LNCS 836*, pages 98–113. Springer, 1994.

[Cau92] D. Caucal. On the Regular Structure of Prefix Rewritings. *Theoretical Computer Science*, 106:61–86, 1992.

[CES86] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications. *ACM Transactions on Programming Languages and Systems*, 8(2), 1986.

[Cou89] B. Courcelle. The Monadic Second-Order Logic of Graphs, II: Infinite Graphs of Bounded Width. *Mathematical Systems Theory*, 21:187–222, 1989.

[Jér93] T. Jéron. Testing for unboundedness of fifo channels. *Theoretical Computer Science*, 113:93–117, 1993.

[MS85] D. Muller and P. Schupp. The Theory of Ends, Pushdown Automata and Second-Order Logic. *Theoretical Computer Science*, 37:51–75, 1985.

Unité de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unité de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unité de recherche INRIA Rhône-Alpes, 46 avenue Félix Viallet, 38031 GRENOBLE Cedex 1
Unité de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unité de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
ISSN 0249-6399